System to catch Implementation gotchas in the RTL Restructuring process

Anmol Rattan, Satinder Malhi, Balwinder Soni
ST Microelectronics, Greater Noida, India
and
Anuj Kumar, Navneet Chaurasia, Sami Akhtar
Synopsys Inc., Noida, India

I. Introduction

In today's world time to market is a very important factor to plan out the development and delivery of customer tailored and cost efficient System-on Chip (SoC) designs so that the requirements of wide range of customers can be met. In order to meet these business objectives, design reuse and derivative designs have become the key components of the today's SoC Design Methodology, which in turn has demanded the quick accommodation of new architectural changes in the existing SoC Designs to meet the desired functional and physical metrics for the next generation SoC designs. Apart from market reasons, some of the design implementation challenges on advanced technology nodes such as physical design changes, low power design implementation, customization of 3rd party IP cores, MBIST/Test logic insertion etc. have also been demanding the moderate to heavy restructuring of an SoC RTL Design. Logical design change requirements also often necessitate the RTL to be restructured. For instance, ASIL-D compliance in Automotive microcontrollers for safety related aspects often necessitate logic like checkers etc to be regrouped into separate LBIST partitions.

Fig.1 lists down some typical design scenarios of hierarchy manipulation and RTL restructuring.

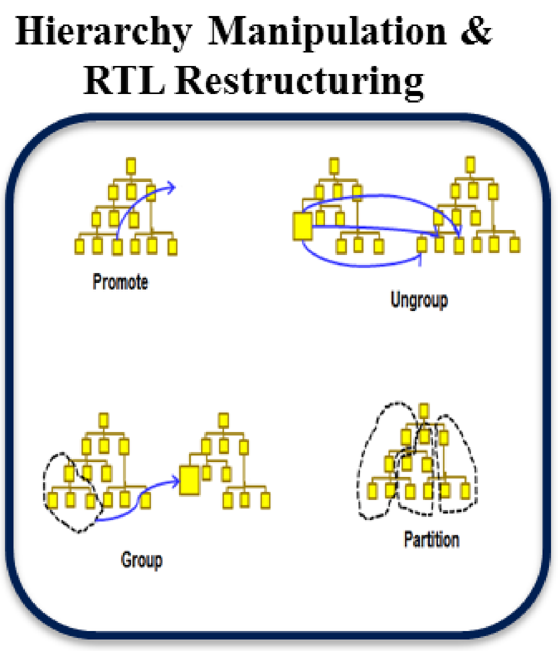


Figure 1. Typical cases of RTL Restructuring

Typically the restructuring of the RTL is either done manually or using home-grown automation scripts/RTL Restructuring and Assembly EDA tool. Irrespective of the approach used, the process of the RTL restructuring is primarily meant to deal with hierarchy manipulations such as group/ungroup/instance movement, wrapper generation and user defined connection stitching, but it does not check for redundant/extra logic, pins and ports created during RTL restructuring process e.g. unconnected pin/ports, wrongly connected pin/ports, propagation of critical design signals like as clock/reset etc. This inadvertently adds undesired design connectivity, logic redundancy issues in the restructured RTL leading to sub-optimal quality of results during design implementation and finally ends up in productivity loss, where long engineering cycles gets wasted out to debug the root cause of these issues in implementation cycle.

Identifying, Debugging, and fixing the issues involves long Turn-Around-Time (TAT) as the later they are identified in the flow, the longer the loop to fix it. When the issues are caught at RTL & Gate-level Verification stage, Formal Functional Equivalence checking, or at the Synthesis stages they involve iterations debug/analysis and exchange across backend Implementation and frontend Design teams. Significant manual effort is involved to debug and fix such issues; thereby causing huge loss of project time, wasted engineering cycles, and ultimately missed deadlines of delivery commitments.

II. PROPOSED SOLUTION

An automated system is proposed using a set of Static checks to catch all undesired design changes leading to connectivity issues upfront at the RTL stage, enabling design implementation to take off smoothly later on. The system targets two versions of the design (RTL), and is not limited to Pre/Post Restructuring – though for clarity and understanding purposes the use case of RTL Pre/Post Restructuring is used in this paper. The proposed flow is depicted below in Fig.2.1.

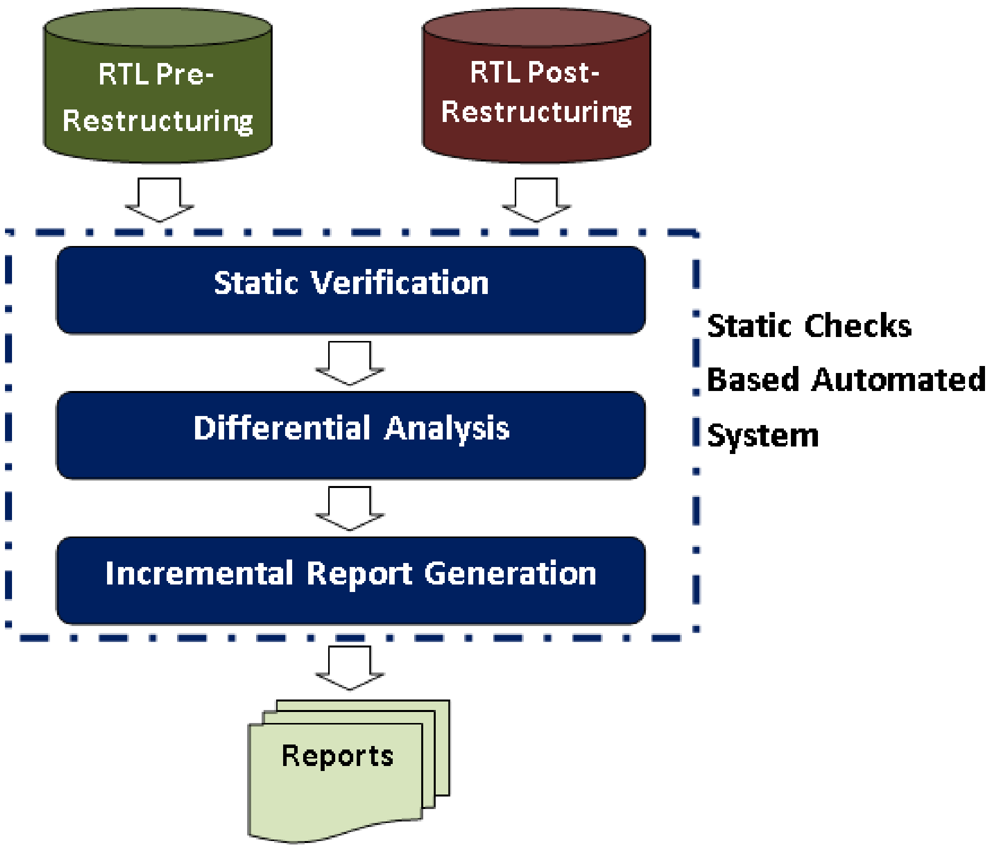


Figure 2.1. Flow Diagram

Flow Description

The Pre-RTL Restructuring design is run through a set of predefined Static verification checks – which identify the potential issues in the design that would impact the design implementation and gate level verification. The generated set of result/information is fed in along with the Post-Restructuring design and run through the same set of checks to generate output such that the issues present is the pre-RTL Restructuring are masked. Thus, only the issues which are induced due to RTL Restructuring are presented in an easy to comprehend manner. Primarily helping identify the incremental issues added due to the Restructuring (change) process.

Running standard full Static checks on a pre-qualified big design (full SOC) would typically flag a huge number (in tens of thousands) of violations – thus making comprehensive checking and analysis undesirable for designers/system integrators already pressed against release schedules. Therefore, a pre-selected set of relevant checks, combined with the differential approach and simplistic presentation of results, significantly brings down the data to analyze and debug - thereby making it an appealing and practically useful solution. Effectively identifying and fixing such critical issues upfront.

Checks are pre-selected from available Static Checks of Lint/CDC/Connectivity solutions in standard EDA industry tools. The selection is based on industry experience (recommendations of experts and prior in-house experiences), anticipation of potential issues, and desired validation of assumptions.

No special setup is required – as it leverages the design setup already available for running any of Static, Synthesis, or Simulation tools. The only requirements are of providing list of RTL files, list of libraries, and constraints for any assumptions. These are standard inputs used by most tools and seamlessly consumable.

List of the main checks used during Static Verification stage of the flow

- Assignments to input ports
- Instances having unconnected/floating ports or hanging nets
- Instances having loaded but undriven inputs/outputs
- Instances having unloaded but driven outputs
- Instances having inputs which are tied to constants
- Non-tristate nets which are multiply driven
- Inout ports that are read but not set
- Width mismatches at the ports/connecting nets
- Configuration mismatch with Specification
- Validating assumptions that require specific values reaching certain nodes/ports – value propagation checks
- Validating clock/reset propagation –
    - Identifying the sequential objects in the design not receiving clocks/reset signals
    - Blocked clock/reset paths
    - Glitch prone clock/reset paths

Report Generation

Reports are generated in standard text format in an easy to comprehend manner. Two level reports are generated – A top-level summary highlighting the violation count and the incremental differences (See Table I), and Secondary level reports one having the details of the incremental violations and another having the problems present in the original RTL.

TABLE I
TOP-LEVEL SUMMARY REPORT

Summary Report for Violation Count			
Rule (Checks)	**Common Issues**	**Only in New RTL**	**Total**
Undriven but loaded input terminal of an instance	3	**1**	4
Inout ports that are read by never set	10	**0**	10
Constants too large. Numeric value exceeds 32-bit capacity	0	**12**	12
Unloaded but driven input port of a module	38	**134**	172

Inout port of an instance in not connected or connected net is hanging	255	**0**	255
Input to a module is tied to a constant Value	0	**1024**	1024
Write to input ports	0	**4**	4
.. ..	.. ..	**.. ..**	.. ..
.. ..	.. ..	**.. ..**	.. ..

III. Results

The flow has been successfully tested on several versions of real multi-million gate Automotive SOC designs (done for 2 full-SOC designs).

TABLE II
Statistics of the Designs

	Design #1	**Design #2**
Gate Count	~15 million NAND2 eq	~17 million NAND2 eq
Std Cell Instance Count	4.45 million instances	4.8 million instances
Flop Count	~659k	~690k
Device Frequency	180MHz	200MHz

TABLE III
Typical Runtimes for the Designs

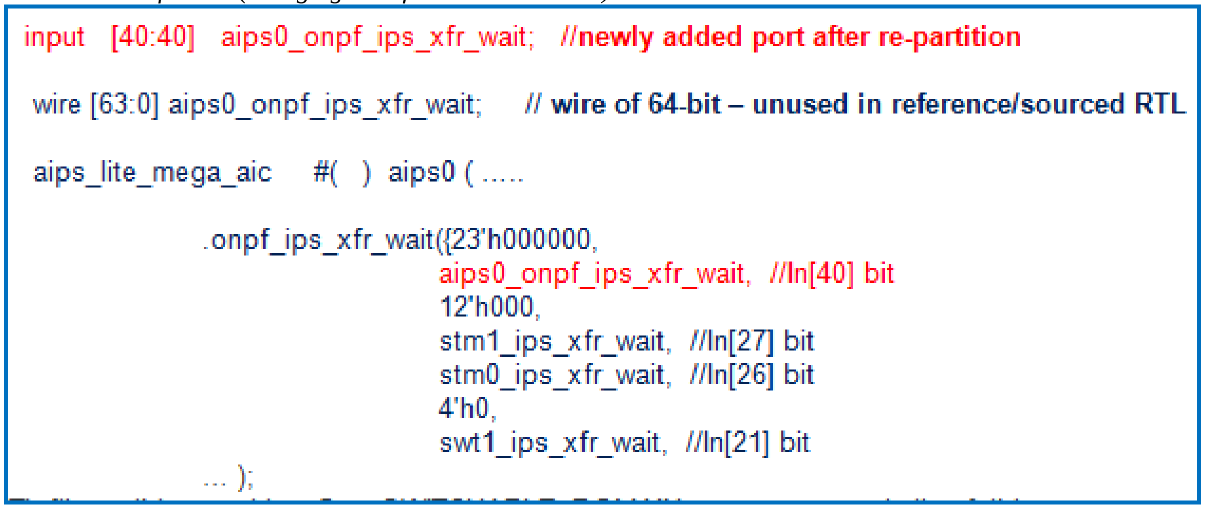
	Design #1	**Design #2**
Design Compilation Rt	50~60 mins	~60 mins
Synthesis Rt	55~58 hrs (~3400 mins)	~55 hrs (~3300 mins)
Formal Functional Equiv Rt	~45 hrs (~2700 mins)	~47 hrs (~2800 mins)
Proposed Static Checks based system	~10 mins	~11 mins

List of design issues, which were identified using this system of static checks on the restricted derivative RTL design

- Mismatch of width in input/wire connections – leading to port Vector or memory size mismatches during synthesis (Fig.3.1)
- Input ports were being written to (wrong assign statements induced)
- Multiple drivers were found (Fig.3.2)
- Multiple Assignments being made to same nets
- Floating inputs/outputs
- Constant Inputs/Nets
- Parsed Parameter values overridden by out of range random/default values
- Buffer insertion in direct port to port connections (Fig.3.3)
- Connectivity issues in the test logic
- Tied low clock-gating logic preventing the propagation of the clock(s) (Fig.3.4)

Here are the details of the exact design issues for some of the violated static checks:

Case#1: Mismatch of widths (wrong signal definitions/connections)



```
input   [40:40]   aips0_onpf_ips_xfr_wait;   //newly added port after re-partition

 wire [63:0] aips0_onpf_ips_xfr_wait;     // wire of 64-bit – unused in reference/sourced RTL

 aips_lite_mega_aic     #(   )  aips0 ( .....

                    .onpf_ips_xfr_wait({23'h000000,
                                        aips0_onpf_ips_xfr_wait,  //In[40] bit
                                        12'h000,
                                        stm1_ips_xfr_wait,  //In[27] bit
                                        stm0_ips_xfr_wait,  //In[26] bit
                                        4'h0,
                                        swt1_ips_xfr_wait,  //In[21] bit
                    ... );
```

Figure 3.1. Width mismatch

Instance “aips0” was moved to another hierarchy. This hierarchy already contained an unused wire of the name “aips0_onpf_ips_xfr_wait” – same as the wire name being used in the connection of the instances’ port: “onpf_ips_xfr_wait” in the earlier hierarchy. Accordingly a port by this name was also punched in the new hierarchy. However, this creates a conflict of width mismatch as the wire and port names match but are of different width definitions. This issue does not get caught until a later stage.

The proposed solution catches such issues upfront with insignificant loss of time and effort.

Case#2: Multiple Drivers created by moving an instance

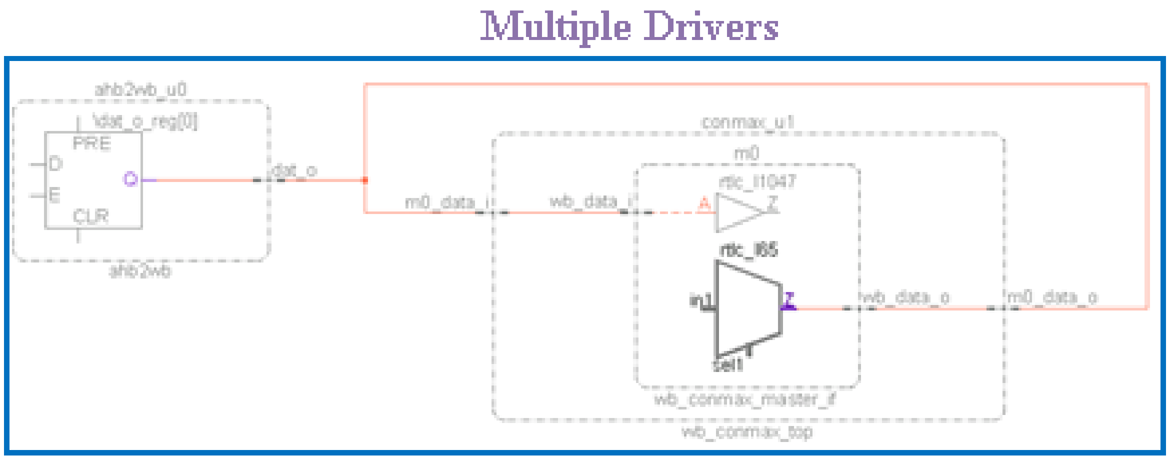
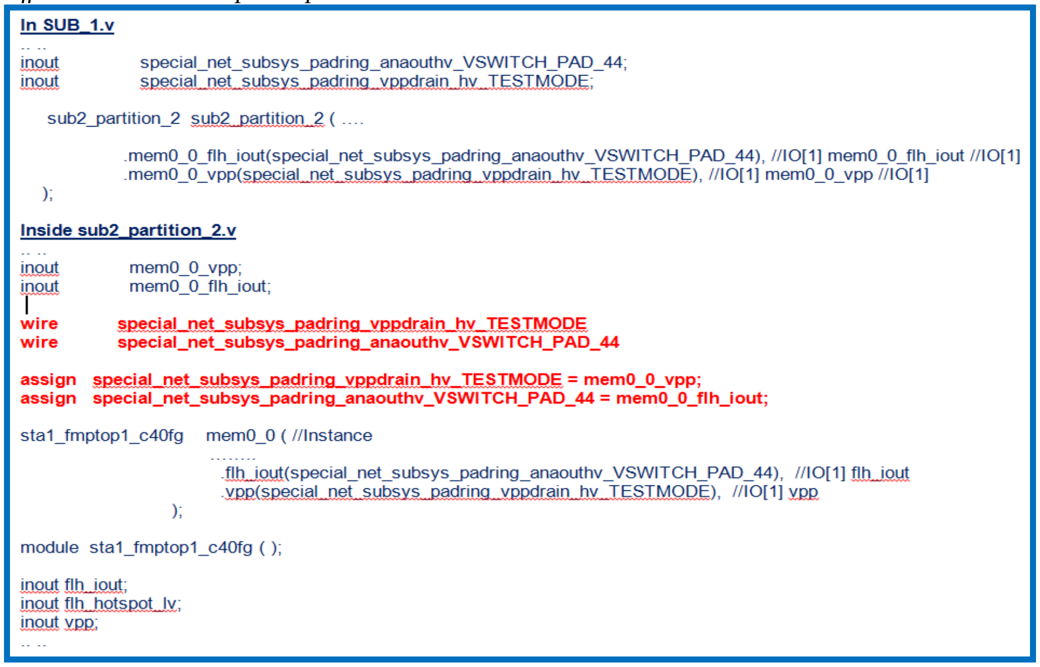


Figure 3.2. Multiple drivers

Case of multiple drivers is also a common problem that results from an incorrect RTL Restructuring or change in the design. These create undesired tri-state buses and might not get caught until a later stage in the design flow. Such cases are also easily identified and flagged by the proposed solution.

Case#3: Buffer insertion in direct port to port connections



```
In SUB_1.v
.. ..
inout          special_net_subsys_padring_anaouthv_VSWITCH_PAD_44;
inout          special_net_subsys_padring_vppdrain_hv_TESTMODE;

   sub2_partition_2  sub2_partition_2 ( ….

            .mem0_0_flh_iout(special_net_subsys_padring_anaouthv_VSWITCH_PAD_44), //IO[1] mem0_0_flh_iout //IO[1]
            .mem0_0_vpp(special_net_subsys_padring_vppdrain_hv_TESTMODE), //IO[1] mem0_0_vpp //IO[1]
   );

Inside sub2_partition_2.v
.. ..
inout         mem0_0_vpp;
inout         mem0_0_flh_iout;
|
wire        special_net_subsys_padring_vppdrain_hv_TESTMODE
wire        special_net_subsys_padring_anaouthv_VSWITCH_PAD_44

assign   special_net_subsys_padring_vppdrain_hv_TESTMODE = mem0_0_vpp;
assign   special_net_subsys_padring_anaouthv_VSWITCH_PAD_44 = mem0_0_flh_iout;

sta1_fmptop1_c40fg    mem0_0 ( //Instance
                         ……..
                          .flh_iout(special_net_subsys_padring_anaouthv_VSWITCH_PAD_44),  //IO[1] flh_iout
                          .vpp(special_net_subsys_padring_vppdrain_hv_TESTMODE),  //IO[1] vpp
                   );

module  sta1_fmptop1_c40fg ( );

inout flh_iout;
inout flh_hotspot_lv;
inout vpp;
.. ..
```

Figure 3.3. Multiple drivers

Instance “mem0_0” was moved inside a parallel instance “sub2_partition_2”. Some ports which were of type “inout” were supposed to be directly connected after punching ports on “sub_partition_2”. However, it happened that instead “assign” statements were inserted for the “inout” port, adding a buffer in the path and effectively removing the “output” path – making it “input” only. This would pose real problems and might not get caught even at the synthesis stage and at a much later stage of simulation or other verification.

These types are easily identified and flagged by the flow – thereby avoiding a big iterative loop.

Case#4: Tied low clock-gating logic preventing the propagation of the clock(s)

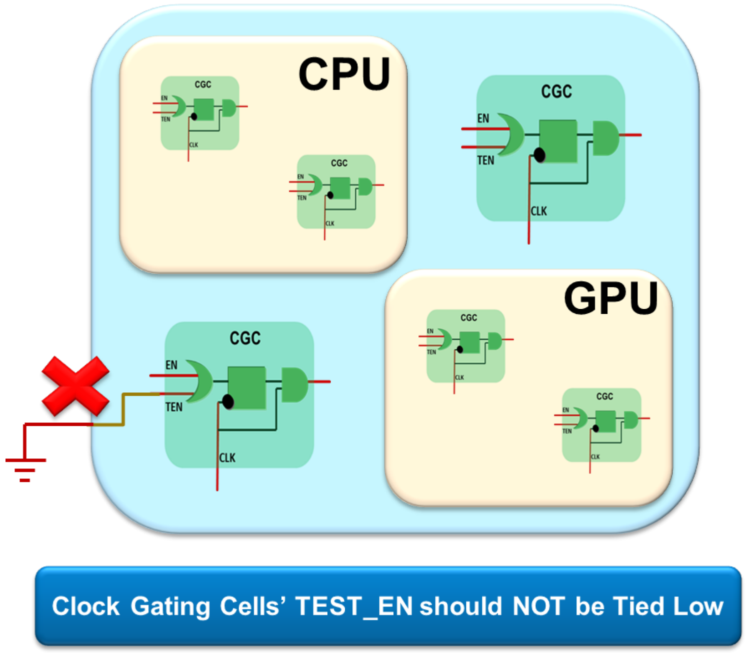


Figure 3.4. Clock Gating Cell (CGC) has enable tied to constant

This is a typical case of checking an assumption that the Clock Gating Cells' (CGC) Test Enable logic is not tied to a constant value. Missing out on such cases can again invoke long iterations as this might get caught later at Functional Equivalence checking. Incase, the earlier RTL also has this problem, it might not get caught until a much later stage – when some simulation pattern for checking the Test Logic might catch it. Such cases could be potentially dangerous and involve an even further bigger loop to close.

Comparison of the Turn-Around-Time (TAT) to Identify/Fix the Issue(s)

TABLE IV
COMPARATIVE TATS

Issue caught at Synthesis stage	1+ weeks (Synthesis Runtime + Exchange across Backend Implementation and Frontend Design teams)
Issue caught at Formal Functional Equivalence check	3+ days (Functional Eq Runtime + Debug/Analysis)
Proposed Static Checks based System	10~20 mins (It uses the same Design Setup)

IV. CONCLUSION

The proposed flow enables frontend design/system integrators to catch potential backend implementation issues early in the design cycle and saves a huge amount of time and iterations from backend to frontend. The achieved runtime of ~10 minutes just highlights the utility of the flow and how it fits in the sign-off flow from frontend to backend without impacting the overall runtimes.

Key highlights of the flow

- Enables designers/system integrators to catch potential backend implementation and RTL Restructuring issues early in the design cycle
- Saves huge amount of time and iterations between backend and frontend design teams – enabling timely delivery schedules
- Increases confidence in the Restructured (changed) RTL and helps the frontend team focus on the job at hand – i.e. SOC Integration
- Automated flow ensures easy integration with the users' current flow
    - requires no special setup or learning curve, thereby easily fitting into the sign-off flow from frontend to backend
    - insignificant overhead in runtime as it runs in a relatively very short time

Thus, the proposed Static Checks based system provides a huge value to chip designers by catching significant issues upfront with an insignificant overhead in terms of runtime and requires no special setup.

Acknowledgment

This research was supported by ST Microelectronics, Automotive Group and Synopsys Global Technical Services Group. We thank the management in both the organization for the support and encouragement during this entire activity.

We thank Sanjum Bhatia, CAE for assistance with trials and data collection for inputs for development of this validation flow.

We thank our colleagues from ST Microelectronics' Automotive Group, Noida – 32-bit Power Train & Safety Microcontroller Design teams - who provided great insight and expertise to foresee problems and shared their experience in various aspects of chip design to help validate the flow and the system.

We thank our colleagues from Synopsys' SpyGlass R&D who provided wonderful support and helped in bringing up and integration of the flow.

Tools and Environment Used

DESIGNS: The flow was validated on 2 real full-SOCs to be used in the Automotive Domain.

STATIC CHECKS: The system was built by using Static checks which are part of the standard checks available in SpyGlass Lint, SpyGlass CDC, and SpyGlass Connectivity Verify solutions.

AUTOMATION: Automation and scripting was done using standard Perl language. Reports were dumped out in standard readable text format.

Authors

ANMOL RATTAN	ST Microelectronics, Greater Noida; anmol.ratan@st.com
SATINDER SINGH MALHI	ST Microelectronics, Greater Noida; satinder.malhi@st.com
BALWINDER SINGH SONI	ST Microelectronics, Greater Noida; balwinder.soni@st.com
ANUJ KUMAR	Synopsys Inc, Noida; anujk@synopsys.com
NAVNEET CHAURASIA	Synopsys Inc, Noida; navneetc@synopsys.com
SAMI AKHTAR	Synopsys Inc, Noida; sakhtar@synopsys.com